The Report committee for Chirag Sakhuja

Certifies that this is the approved version of the following report:

Creating and Teaching Introduction to Linux as a Master's Student

APPROVED BY

SUPERVISING COMMITTEE:

Jonathan Valvano, Supervisor

Christine Julien

Creating and Teaching Introduction to Linux as a Master's Student

by

Chirag Sakhuja, B.S.E.E.; B.S.C.S.

REPORT

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ENGINEERING

THE UNIVERSITY OF TEXAS AT AUSTIN

May 2017

Creating and Teaching Introduction to Linux as a Master's Student

Chirag Sakhuja, M.S.E.
The University of Texas at Austin, 2017

Supervisor: Jonathan Valvano

In this report, I discuss my experience creating and teaching a course called Introduction to Linux. The course provides students with enough knowledge to feel comfortable with a command line in the Linux operating system. Introduction to Linux is geared toward underclassmen in the ECE department at The Univeristy of Texas at Austin, though it is open to anyone intersted in becoming familiar with Linux. I describe the motivation for creating the course, the methodology I emplyed when teaching the class, and steps that can be taken to improve the course in future semesters.

Table of Contents

Chapter 1

Introduction

In the Spring 2017 semester, I created and taught a course called Introduction to Linux in the ECE department at The University of Texas at Austin. The purpose of the course was to teach students the fundamentals of using a command line in Linux, with a focus on commands that students would find useful in an industry internship. While targeted toward first and second year students, the course was open to anyone wanting to learn about Linux.

The course was unprecedented in the ECE department for two reasons: it was taught by a Master's student and it was a single credit hour, credit/no-credit course that lasted for the first six weeks of the semester. The format of the course allowed students to learn valuable skills by adding a light workload to the first half of their semesters.

In the following sections of this chapter, I will explain why Linux is advantageous and motivate my reason for creating and teaching the course. In successive chapters, I will discuss my teaching methodology and future steps.

1.1 The Advantage of Using Linux

Linux is an operating system that is in use in many electrical and computer engineering companies. By design, Linux provides a large set of single-purpose commands that can be combined to complete complex tasks. Possible tasks may be processing documents into spreadsheets to analyze their data or managing the files in a large collaborative project. Commands generally serve a single purpose and provide several flags to modify their behavior. One example of such a command is `tr`, which modifies a string of characters by deleting all instances of a character, squeezing contiguous instances of a character, or converting one character to another. On its own, `tr` can perform a limited set of operations. However combined with a suite of text processing commands (`cut`, `sed`, etc.), it can be used to process documents.

Often, the commands are available for usage in two forms: a command line interface (CLI) and a graphical user interface (GUI). The GUI is what most students are familiar with before entering the course. Microsoft Windows and macOS both rely heavily on applications with a GUI. The CLI, sometimes referred to as the terminal, is usually more challenging to use but allows for greater flexibility. Commands may be chained together using the CLI, but not the GUI.

Due to the variety of commands and parameters, there is often more than one way of completing a given task in Linux. Thus, it is sufficient to understand a subset of the commands available and learn how to combine them. I take this approach in my teaching style, as discussed in Section 2.2.

1.2 Motivation

Linux has become the pervasive operating system in industries related to electrical and computer engineering. However, the ECE department at The University of Texas does not provide any formal education in using Linux. It is not difficult for students to pick up commands as needed, but doing so may result in the students learning a hodgepodge of commands without understanding their proper usage. In my own experience, I have seen students memorize a series of commands to achieve a task, only to be completely lost when one of the steps does not do what they expect.

According to several personal accounts, students that are required to learn how to use Linux during an internship often feel overwhelmed for the first several weeks. These students learn just enough to get their work done, but do not understand what the purpose of the commands are necessarily. As mentioned previously, learning commands on an as-needed basis from the beginning makes it difficult to grasp the underlying principles of the commands. Furthermore, learning how to work with an operating system detracts from the amount of time a student can focus on their engineering work.

Introduction to Linux addresses both of these issues. The course is designed to introduce students to Linux early in their education so that they feel comfortable with it throughout their academic and professional careers.

I address the first issue of the department lacking a course by having a graduate student lead it. While the faculty in ECE almost unanimously agree

that this course is necessary, it is difficult for a professor to dedicate their time to this course when they could be teaching a more substantial engineering course. It is useful, then, to have the course be lead by a graduate student under the guise of a Teaching Assistantship. To ensure the course follows university policy, a professor must be present at the lectures.

I address the second issue of introducing Linux to students early on by gearing the course toward first and second year students. In the Spring 2017 semester, the Registrar initially restricted registration to first year students and gradually allowed other students to register. The hope is that Introduction to Linux becomes a course that most students take in their early semesters so that future courses may use Linux without wasting time instructing on how to use it.

Chapter 2

Course Structure

When designing the course structure, including lecturing style, assignments, and lab sections, I focused on achieving two primary goals: encouraging students to explore commands and flags on their own and keeping students with various backgrounds engaged. Since Linux commands perform specific, small tasks, it is more important for students to understand what the commands are capable of rather than memorizing the syntax and behaviors of the commands. Furthermore, Linux commands are best learned through practice, so students must experiment on their own.

In the following sections, I describe how I taught lectures and lab sections as well as explain the motivation behind the assignments. For details about the lectures and assignment, please refer to the syllabus in Appendix A.

2.1 Logistics

The course was listed as credit/no-credit by the department and there was one lecture and two lab sections per week, for the first six weeks of the semester. Attendance was not required in either the lectures or the lab sections. Some of the assignments were graded based on completion while others

were based on correctness.

To receive credit for the course, the student must have received at least a 50% on their assignments after dropping up to two assignments. I was fairly lax on the deadlines for assignments. I would often extend deadlines so that the students would have some motivation to continue trying without feeling pressured.

2.2 Lectures

For each lecture, I introduced a command or series of related commands, provided an interactive demo, and then asked the students to complete an exercise. Introducing a command involved going through the command's man page and selecting what I thought were the most useful flags. After going over the basic usage, I would demonstrate how the command could be combined with other commands. Then, I would give students a scenario in which the command would be useful and let them spend a few minutes talking amongst each other to figure out the correct usage.

I found that during the exercises, students would raise questions they hadn't thought about after my explanation of the command. The questions would be over some flags or some alternative command to achieve the same goal. This directly aligned with the goal of helping students understand the purpose of the commands.

For a full description of the commands covered in lecture, see Appendix

A.

2.3 Lab Sections

Lab sections augmented lectures by covering the same commands as in lecture but introducing more flags. Lab sections consisted of giving the students many more exercises than in lecture and letting them work with the rest of the class. The discussion helped students stay motivated and see multiple solutions. I would often give exercises that required students to search through man pages or forums to figure out the best solution. This, once again, helps the students understand what the commands are capable of rather than memorize what was being taught in lecture.

2.4 Assignments

The assignments were open ended such that the task was simple to achieve using a handful of commands, but could be made much more complicated. One assignment involved compiling a Nintento® GameCube and Wii emulator called Dolphin. Compiling Dolphin requires installing several packages, most of which are uncovered in the middle of compilation. While somewhat time consuming, the dependencies were not overly difficult to install. However, Dolphin provides many optional features that require additional packages to be present to compile. A student may choose to install the minimum set of packages or they may choose to track down every feature, flag, and package necessary to enable all optional features. The latter requires a

more thorough understanding of the build system, and thus takes more time, but both options would get full credit on the assignment.

Each assignment was also intended to model a real world scenario. The assignment on extracting hyperlinks from an arbitrary webpage was similar to a task I was assigned at one of my internships. Similarly, one assignment required students to navigate through files using git, a system that manages versions of files, using a series of commands that are very applicable to any collaborative project.

For a full description of the assignments, see Appendix B. Assignment 1 was to show up to the lab section of the first week so the students would have an idea of what to expect.

Chapter 3

Future of the Course

Spring 2017 was the first semester Introduction to Linux was offered. The ECE department and faculty have expressed interest in continuing the course, so it is important to discuss what changes should be made in future semesters. I collected feedback and have recommendations on how to improve the course in future offerings.

3.1 Feedback

Due to the experimental nature of this course, collecting feedback was crucial. Throughout the semester, I asked the students how I could improve the lectures. Some of the feedback could be applied immediately, such as creating a quick reference sheet with a brief summary of how to use the commands we covered (see Appendix C). Halfway through the semester, I conducted an informal survey of students' thoughts. I also conducted course instructor surveys (CIS) as per university policy, though these results have not been returned at the time of writing this report.

Overall, the feedback was strongly positive. Students appreciated the style of the lectures, stating opinions such as "this [course] sets up a perfect

learning environment." Students also gave feedback saying they may not immediately understand the value of some of the commands we covered, namely the text processing commands, but they are confident the commands will be beneficial in the future. Additionally, some students have already found some commands to be useful and have incorporated them into their schoolwork.

The most common complaint about the course was that it covered commands that the students never saw on the assignments. This resulted in students forgetting the purpose or usage of a command. Another complaint was that the course moved too quickly without lectures building upon one another. I remedied these two complaints somewhat by creating the quick reference sheet, but the lectures could be reorganized.

3.2 Improvements for the Future

While overall a success, Introduction to Linux can be improved for future semesters. Some of the lectures covered several commands, many of which are fairly complex. For example, `vim` and `tmux` are very powerful tools to improve productivity, but each has a steep learning curve. I covered both of these in a single lecture, in addition to `ssh`.

To address the issue of students forgetting some commands, the assignments can be made a little more complex. In particular, the assignment after the text processing lecture covered only a couple of commands we covered in lecture.

Finally, for the assignment to compile the Dolphin emulator, students were allowed to work with a partner. Students enjoyed working with a partner and most students finished this assignment earlier than the others.

Appendices

Appendix A

Syllabus

Course Description

Linux and tools associated with Linux are prevalent throughout industry from software development to analog IC design. Understanding how to use Linux effectively can improve productivity substantially. In this course we will introduce you to the concepts behind Linux systems and show you how to use some of the fundamental tools. This is by no means a comprehensive course on how to use Linux. Instead we will equip you with the skills to understand what Linux, particularly a Linux shell, is capable of.
Prerequisites: None.
Target Audience: Students who have no experience with Linux. No programming knowledge is required, but it may be helpful. Undergraduate status is required.
Instructor: Dr. Christine Julien
Contact: c.julien@utexas.edu
Office Hours: T 11am-12:30pm and Th 8-9:15am in POB 5.140

Course Objectives

At the completion of this course, students will be able to:

1. Comfortably use a Linux command line.
2. Combine Linux tools to perform complicated tasks.
3. Improve productivity by taking advantage of existing tools.

Class Meeting Times

Lectures will be held on Tuesdays from 3:30-5pm. Class will be held for only the first six weeks of the semester (1/17, 1/24, 1/31, 2/7, 2/14, and 2/21). The course also involves a lab section, which will be either Wednesday or Thursday from 3:30-5pm (depending on which section a student is registered for) for those same weeks.

Course Logistics

There will be one 1.5-hour long lecture and one 1.5-hour long lab session per week for six weeks total. There will also be weekly assignments (a total of 6 assignments). The course is pass/fail, which will be determined by the completion of the assignments. To pass, you must receive an assignment average of at least 50% and you may miss at most two assignments. Attendance to lecture and lab is encouraged, but not mandatory, **other than the first lab session, which counts as the first assignment**.

Advising Notes

This course can only be taken on a pass/fail basis and cannot count towards the academic enrichment technical core. More specifically, the EE 107S course does not count towards the ECE major, and has 0 hours of engineering topics and 0 hours of math and basic science for the purposes of ABET metrics.

Lectures

Each lecture, we will pick a single problem and work our way to a solution while explaining the necessary tools along the way. Some examples are:

- Downloading an open-source project and going through the process of trying to compile it and fixing the inevitable errors and dependencies so that the project can be installed.
- Searching through and parsing a run log to extract data that can be plotted.

The lectures will be interactive, so please bring your laptops. Additionally, there will be assignments after each lecture that build upon the concepts taught that week.

Tentative Course Outline

Week	Content
Week 1	• Course outline and overview • What is Linux and how is it used? • Introduction to shells (we will be using Bash) • File system navigation and structure (`ls`, `pwd`, `cd`, `mkdir`, `rm`, `cp`, `mv`, `find`) • **Assignment:** Look through man pages for each of the commands covered; attend the first lab session!
Week 2	• Viewing files (`cat`, `head/tail`, `less`, `tee`) • Pipes and I/O redirection • File searching and processing (`grep`, `tr`, `cut`, `sed`, `awk`) • Introduction to Bash scripting • **Assignment:** Generate a summary from a log file
Week 3	• Editing text in the command line (`nano`, `vim`) • Managing multiple shells (`tmux`) • Customizing tools with dotfiles • Accessing remote machines (`ssh`, `scp`) • **Assignment:** Create a personalized .vimrc and .tmux.conf
Week 4	• File compression and security (`zip`, `tar`, `chmod`, `sudo`) • Package managers and installing new software (`apt`) • Building and installing from source (`make`, configure scripts) • **Assignment:** Install a buggy open-source project and all of its dependencies without sudo
Week 5	• Introduction to version control • Basics of `git` • **Assignment:** https://github.com/git-game/git-game
Week 6	• Advanced Bash scripting • Managing multiple jobs (`jobs`, `bg`, `fg`) • Managing processes (`ps`, `htop`, `kill`) • Miscellaneous topics (networking, debugging, etc.) • **Assignment:** Create a script that runs multiple instances of a program in parallel with different command line arguments

Appendix B

Assignments

Assignment 2

In lecture and lab section we went over some of the text processing commands available on Linux. We went through some basic examples, as well as some more advanced examples. However, many of the concepts and commands covered are very expansive. The goal of this assignment is to give you a strong grasp of how to use what we learned.

Bash scripts

So far we have been referring to the command line as the terminal. However, a terminal really just refers to the window in which you enter commands. The terminal window contains a shell inside it, which is really what the command line is. Bash is a shell that launches inside the terminal window by default on Ubuntu 16.04.1 LTS. Other shells include `zsh` and `tcsh`, and these can be set to run in the terminal window by default instead. The main difference between shells is the syntax for writing shell scripts. Note that shell scripts are not to be confused with the command `script`.

Shell scripts contain a sequence of statements to execute. Shell scripts are similar to programs written in Python, Ruby, Perl, etc. We will discuss some of the programming constructs in Bash scripts (i.e. shell scripts written for the Bash shell) in a later lecture. For this assignment, all you need to know is that you can specify a list of commands in a shell script that will be executed in sequential order.

Given below is a short Bash script that lists out the files and directories in the current directory.

```
#!/bin/bash

echo 'Writing file names to files.txt'
ls -l | tr -s ' ' | cut -d ' ' -f 9 > files_temp.txt
tail -n +2 files_temp.txt > files.txt
rm files_temp.txt
```

The following is a more detailed explanation of the Bash script.

- `#!/bin/bash` is a special line that all Bash scripts will start with. It indicates to the shell that the file should be treated like a Bash script.
- Line 3 echoes a message to the user. `echo` is very useful in shell script settings to give users information on what the script is doing or if there have been any errors.
- Line 4 performs a directory listing (`ls`) and then extracts the 9th column. The 9th column contains the names of the files and directories. The output of this entire pipeline of commands is written to `files_temp.txt`.
- Line 5 removes the first line from `files_temp.txt`, because it ends up being an empty line, and outputs the rest of the lines to `files.txt`. Note that we haven't seen the `+2` notation before, but it is explained in the `man` pages.
- Line 6 cleans up the temporary file.

This shell script can be written as a single pipeline of commands, but this form exemplifies a powerful way of using shell scripts. In a shell script you can run multiple unrelated commands and output the result of each into a separate file. Then you can run a sequence of commands that assembles all the temporary files into a single final output. Finally, you can clean up all the intermediate files.

Running a shell script

You may write a shell script using the visual text editor in Ubuntu 16.04.1, `gedit`. Once you have written the file, you must make it executable. You only need to make a file executable one time. To make the file `shell.sh` executable, you can run the command `chmod +x shell.sh` (we will discuss this in a later lecture).

Now that the file is marked as executable, it is ready to be run. To run the file `shell.sh`, you can type in `./shell.sh` and then press Enter. The `./` means to look for the `shell.sh` in the current directory. The reason you must explicitly state the directory of the script will be explained in the future.

Assignment

Your job is to write a Bash script that processes the hyperlinks in a text-only version of a webpage.

How to use the starter files

You are given a text-only version of `https://www.utexas.edu/` to experiment with when writing your Bash script. You are also given a file that shows you the expected output of your Bash script. Both of these files can be downloaded by running the following commands:

```
curl http://a-dev.me/107s/webpage.txt > webpage.txt
curl http://a-dev.me/107s/expected_output.txt > expected_output.txt
```

You should notice that all of the content that shows up on the visual version of the webpage (which you would view in a browser) also shows up in the text file, although it is unsurprisingly very difficult to read. We will just be concerned with the hyperlinks that show up on the webpage.

In the text-only webpage, hyperlinks are denoted by a number within square brackets (e.g. [38]) followed by the text you would normally click on in a browser. One example is the link to "Subscribe to our newsletter", which shows up at the very bottom of the webpage in a browser. At the time of writing this document, this link corresponds to "[113]Subscribe to our newsletter" in the text-only webpage.

At the bottom of the text-only webpage, the URL for each hyperlink number is given in two sections: Visible links and Hidden links. For each hyperlink with a number in the body of the webpage, there is a corresponding URL in the Visible links section. Additionally, there are some hyperlinks on the page that do not have text that you can click on in a browser. These hyperlinks are given under the Hidden links section.

Expected behavior

Given the text-only webpage, your job is to produce a summary of the hyperlinks. For each hyperlink, your Bash script should output a single line. Each line should start with the hyperlink number in square brackets, followed by the rest of the line in the text-only webpage, followed by a TAB character, followed by the URL that the hyperlink corresponds to. In the `webpage.txt` file, you will see the following:

```
   #[1]The University of Texas at Austin RSS

   [2]Skip to main content
   ...
References

   Visible links
   1. https://www.utexas.edu/rss.xml
   2. https://www.utexas.edu/#ut-page-content
   ...
```

Your Bash script should output the following:

```
[1]The University of Texas RSS    https://www.utexas.edu/rss.xml
[2]Skip to main content https://www.utexas.edu/#ut-page-content
...
```

A few things to note in the output.

- Only lines with hyperlinks are shown in the output.
- Every line must start with a [. In other words, you must remove everything from the beginning of the line until the hyperlink number.
- After "RSS" on line 1, there is a TAB character. Similarly, after "content" on line 2, there is a TAB character. The TAB characters end up not aligning properly, but that is fine.
- In the `webpage.txt` file, the Visible links and the Hidden links start with a hyperlink number, followed by a period, followed by a URL. You should only keep the URL column.
- Since there are no corresponding hyperlinks in the body of the webpage for Hidden links, you may just tack on all the hidden links at the end of the file (you should not need to do anything extra to achieve this behavior). If you look at the last 3 lines in `expected_output.txt`, they just contain a TAB and then the Hidden link.

For more examples, you can look through the `webpage.txt` and `expected_output.txt` files.

We are giving you a sample text-only webpage, but when grading we may use other webpages that have a similar structure. We will guarantee that there will be at most one hyperlink per line.

Important notes

- You can view the files using the GUI of your virtual machine. If you double-click the text files in the file browser, it should open up a text editor called `gedit`.
- Your Bash script must always expect the text-only webpage to be in a file called `webpage.txt`. The final output file must always be named `output.txt`. Your Bash script must be called `get_links.sh`.
- You should leverage writing temporary files in the Bash script and then merging them together into a single file at the very end. The `paste` command will prove to be useful to combine files together line by line, and it combines files using a TAB character by default. You should also clean up all temporary files using the `rm` command at the end of the Bash script.
- The '^' character is a special character in regular expressions that matches the beginning of a line. For example, if you want to remove the leading spaces of each line, you can use the command `sed 's/^[ ]*//' file`. This command will match against

spaces starting at the beginning of the line and then replace them with nothing, thereby removing all leading spaces. If you run the command `echo '  Hello, world!' | sed 's/^[ ]\+//g'`, the output will be "Hello, world!" instead of " Hello, world!", despite the g modifier that would usually match other spaces in the line. Notice that only the leading spaces have been removed.

- For those of you interested, you can create text-only webpages using the `lynx` command. `lynx` is a browser that runs entirely out of a terminal and only displays the text components of a website. To create the text-only webpage, I ran the command `lynx --dump https://www.utexas.edu/`.

- You may find the `diff` command to be useful. `diff` will compare two files and show you any lines that differ. If the files are exactly the same, `diff` will not produce any output. For example, `diff output.txt output.txt` will not have output. To compare your output file with the expected output, you can run the command `diff output.txt expected_output.txt`.

- I was able to come up with a solution that used a single `paste`, `grep`, `sed`, and `awk` command each with fairly straightforward arguments. It may be difficult to arrive at the solution without much experience, but don't overthink it too much! Also, you are free to use any number of any command available with the default installation of Ubuntu 16.04.1 LTS. If you want to use a single `awk` command, be my guest.

Assignment 3

With Vim (or Emacs) and Tmux it is possible to move your workflow entirely to a terminal. Both tools provide a lot of functionality and massive configuration options. For this assignment you will set up a `.vimrc` (Vim configuration) and a `.tmux.conf` (Tmux configuration).

Assignment

Create a `.vimrc` and a `.tmux.conf` that meet the requirements below. Both files should be in your home directory (`~`). If the files do not exist, you should create them. It's a good opportunity to learn Vim while writing these files!

Setting up `.vimrc`

We talked about some commands in Vim that are prefaced with a colon. One example is search and replace using `:%s/regex/replacement/g`. There are many commands in Vim that allow you to customize behavior as well. For example, you may change the color scheme by using the command `:colorscheme elflord`. However, you may notice that if you exit Vim and relaunch it, all your customization is removed. In other words, colon commands (also called Ex commands) only affect Vim for the open session.

To get around this, there is a special file called `.vimrc` that Vim goes through every time it launches. The contents of the `.vimrc` are just a sequence of Ex commands without the colon prefix. Shown below is a very simple `.vimrc` that just sets the color scheme to `elflord` in Vim every time it is launched.

```
colorscheme elflord
```

Vim also supports plugins and plugin managers. Currently a very popular plugin manager is Vundle. Vundle uses Github (a popular site for sharing open source projects) to download plugins automatically. This makes it really easy to set up a `.vimrc` on one machine and then transfer it to another to easily set up Vim on that machine exactly the same way.

You will make and submit a `.vimrc`. You should customize Vim to your heart's desire, but you must at least have the following:

- A different color scheme than the default
- The plugin manager Vundle
- At least one plugin of your choice (plugins you use for a color scheme do not count)

My `.vimrc` can be accessed here. I like the color scheme Molokai and only use a small number of plugins. Of my plugins, NERD Commenter is probably the most immediately

useful, especially if you are doing any sort of software development in Vim. You can complete this assignment just by copying my `.vimrc`, but that's not the point! You should try to familiarize yourself with Vim and and see exactly what you want. Most likely, your `.vimrc` will evolve a lot as you continue to use Vim. Here are a couple other references that should give you a good starting point.

- Janus is a project that quickly improves Vim with a bunch of intuitive settings and plugins. It's overkill in a lot of situations and may slow Vim down considerably, but it's a really good starting point. Disclaimer: it's supposed to be easy to install, but in my experience I've had to install lots of other things for it to work.
- Vim Awesome is a site that lists a bunch of plugins by popularity. In true Vim fashion, you can also navigate the home page using keyboard shortcuts alone.

Note: Vundle relies on a tool called `git` that we will talk about later. Ubuntu 16.04.1 LTS doesn't include `git` out of the box, so you must run the command `sudo apt install git` *before* setting up Vundle.

Setting up `.tmux.conf`

By default, Tmux is very usable. While Tmux does have the semblance of plugin support, Tmux plugins are not as powerful or necessary as Vim plugins. Instead, you will focus on modifying the appearance of Tmux. You should create a `.tmux.conf` that meets the following requirements:

- Change the prefix from ctrl+b to something else
- Change the color scheme to match Vim (Hint: you may find tmuxline.vim to be a useful plugin)

My `.tmux.conf` can be accessed here. I have some tweaks that improve Tmux portability and make Tmux navigation similar to Vim navigation. I also have a color scheme defined in a separate file called `.tmux.colors` that my `.tmux.conf` loads. My `.tmux.colors` can be accessed here.

Submission

Please turn in your `.vimrc` and `.tmux.conf` files on Canvas without renaming them. You may use `scp` to transfer your files to your host machine and then submit from there. If you would like to submit through Firefox in your virtual machine, you will need to show the "hidden files" in your file browser. By default, when you are trying to select files to upload, dotfiles do not show up. When the file explorer window pops up, you can press ctrl+h to toggle visibility of dotfiles.

Assignment 4

The purpose of this assignment is to familiarize yourself with building and installing a complicated project from source. The assignment itself may seem straightforward, but you'll find that you will need to install many non-obvious dependencies to successfully build the project. This is common when working on any large open source project.

For this assignment you may work with up to one partner.

Assignment

Dolphin is a cross-platform, open source, GameCube and Wii emulator. Your job is to build and install Dolphin from source. First, download the Dolphin source code using the command:

```
git clone https://github.com/dolphin-emu/dolphin.git
```

Then follow the build instructions for a Linux Global Build that are reproduced (slightly modified) below.

```
mkdir build
cd build
cmake ..
make -j4              ; build with 4 threads
sudo make install
dolphin-emu
```

The most immediate dependency is a program called CMake. CMake is a build system that generates a `Makefile`. One of the steps CMake will perform is to make sure all dependencies necessary are installed. You will spend most of your time making it through the `cmake ..` command. Once you have built and installed the project, you may use the command `dolphin-emu` to launch Dolphin.

And...that's it!

Important notes

- Building Dolphin requires at least 1 GB of free space. You can run the command `df -h .` to see how much space is available on the current partition. If your virtual machine does not have enough space, email one of the TAs for instructions.
- Building Dolphin requires many dependencies to be installed. If you do not have `sudo` access (e.g. if you are using the LRC machines), you should work with a partner who does have `sudo` access.

- Since Dolphin is a large project, it will take some time to build (for reference, it took me 10 minutes with an i5-6600K @ 4.6 GHz with 4 physical cores dedicated to the virtual machine). Luckily, CMake generates a Makefile that shows the percentage complete after each individual file is compiled. I would recommend working on the more powerful of the two machines between partners.

- It is often difficult to understand the errors that CMake or `make` will generate. It may be helpful to search some of the words in the error message and then tack on "dev tools ubuntu". For example, you may search "x11 dev tools ubuntu" during this assignment.

- There are lots of additional features you can build into Dolphin if you want (you may notice some dependencies are skipped along the way and features are disabled instead of causing an error). However, for this assignment you just need to be able to get through the basic build completely.

Submission

Instead of submitting a script for this project, you will submit a report describing the steps necessary to build and install Dolphin. The report can simply be a list of packages you had to install along the way. If you've forgotten the packages you installed, you may type in the command `history | grep 'apt install'`. In the report you may also describe packages you had particular difficulty installing or anything else you learned along the way. Submit one report per group to Canvas and include both partners' names on it.

Assignment 5

In this assignment you will practice your Git skills and see how Git can be useful to navigate versions.

Assignment

The assignment is simply to walk through this game. The game is structured as a series of levels, where you must execute one or more `git` commands to move to the next level. The directions for each level are usually specified in the `README.md` file. Remember, since you are moving between versions, each `git` command may change the `README.md` file.

You can go as far as you want, but I would recommend at least trying to get through level 8.

Submission

Once again, you will submit a report for this assignment. The report must at least contain the `git` commands you used to get through each level. You may include any extra information, such as specific difficulties you had or interesting things you learned along the way.

Assignment 6

In this assignment you will learn some basic process management and Bash scripting.

Assignment

The assignment is to create a Bash script that will launch multiple instances of a program we wrote, `runme`, in parallel. The program takes 2 seconds to run, and our goal is to make multiple instances of this program run in approximately 2 seconds as well. Furthermore, we want to redirect the outputs of each of the instances to individual files. Here are the steps you may follow to build up your solution.

Running `runme`

To build the executable, simply type `make`. This will generate an executable called `runme`. To use the executable, you must supply an ID number. This can be any number. For example, `./runme 1` is a valid execution. Note that the output contains the ID number that was given as well as how long the program took to execute. The program doesn't do much else other than sleep for 2 seconds and then output the execution time. Execution time should be very close to 2 seconds.

Sequential `batch.sh`

Now you will write a script that runs instances of `runme` sequentially. The `batch.sh` script should take a single argument that specifies the number of instances of `runme` to run. If an incorrect number of arguments is supplied, you should output the following message exactly and exit with error code 1:

```
Usage: ./batch.sh num
```

If exactly 1 argument is supplied, then you should continue the script like normal. For each instance of `runme` that the script executes, you should output a single file that contains the output of that instance. Recall that to redirect the output of an executable, you can use the `>` character. For example, to redirect the output of a single instance of `runme`, you can run the command `./runme 2 > log2.txt`.

The name of the output files should be given as `log<ID>.txt`, where `<ID>` is the ID argument of the `runme` instance. Thus, if you run `./batch.sh 5`, you should expect the script to create 5 files: `log1.txt`, `log2.txt`, `log3.txt`, `log4.txt`, and `log5.txt`. The contents of `log4.txt` should look similar to:

```
Starting process 4
Execution time: 2002.88 ms
```

The other logs will look similar, except the ID will be different.

Note: Remember that your Bash script must begin with the line `#!/bin/bash`, and you must run `chmod +x batch.sh` once before you can execute it.

Parallelizing and timing `batch.sh`

Our goal is to run multiple instances of `runme` concurrently. It is fairly simple to change a sequential version to be parallel in Bash, but first we'll look at how to time the execution of a command.

To see how long the execution of the entire `batch.sh` script takes, you can use the `time` command. The `time` command allows you to supply a command and then it outputs the amount of time it took to run the command. If you run the command `time ./batch.sh 10`, the output of the sequential version of `batch.sh` will look similar to:

```
./batch.sh 10  0.02s user 0.02s system 2% cpu 20.109 total
```

The relevant number is the last one, `20.109`, which refers to the seconds of execution time. Since we are running 10 `runme` instances sequentially, the execution will be roughly $10 * 2 = 20$ seconds. This is because we must wait for each `runme` instance to fully complete after 2 seconds before moving to the next one.

To parallelize a sequential version of `batch.sh`, you just need to move the `runme` calls to the background using `&`. Once you do this, if you notice that the `time` command is outputting a total execution time of less than 2 seconds, then you may need to look into the `wait` command in a Bash script.

When you have parallelized `batch.sh` and you run the command `time ./batch.sh 10`, you should expect the output of time to look similar to:

```
./batch.sh 10  0.02s user 0.02s system 2% cpu 2.030 total
```

Since all instances of `runme` are executing in parallel, even though each one takes 2 seconds to execute, the overall time only marginally increases past 2 seconds.

Submission

You should just submit the `batch.sh` file, with that exact name, to Canvas.

Appendix C

Quick Reference Sheet

Disclaimer: This reference sheet is not meant to be comprehensive. It gives a quick review of the commands we have covered in lecture and how we used them.

Notation

Each section header is the command and its corresponding usage. Parameters that are optional are in square brackets (e.g. `[OPTION]`). If multiple parameters may be specified, the parameter is proceeded by `...` (e.g. `FILE...`).

Basics of Linux

`ls [OPTION]... [FILE]...`

Displays the file listing in each of `[FILE]...`

Command	Behavior
`ls` `ls .`	Display the contents of the current directory.
`ls DirA`	Display the contents of the `DirA` directory.
`ls -l`	Display more information about the contents of the current directory (file permissions, last modified, ...).
`ls -l -h` `ls -lh`	Same as `ls -l`, but shows the file sizes in human readable formats (e.g. 4K instead of 4096).
`ls -l -t` `ls -lt`	Same as `ls -l`, but sorts the listing by decreasing date modified .

`man page`

Gives a detailed description of the command specified by `page`. The page is scrollable using the arrow keys. To escape, press `q`. To search, press `/` and then start typing the regular

expression search query. To find the next instance of the search, press `n`, and to find the previous instance of the search, press `N`.

Command	Behavior
`man ls`	Open the manual page for `ls`.

`pwd`

Shows the full path of the current working directory (CWD).

Command	Behavior
`pwd`	Display the CWD.

`cd [DIRECTORY]`

Changes the current working directory (CWD) to `DIRECTORY`. `DIRECTORY` can be an absolute path (e.g.
`/home/users/chirag/DirA`) or a relative path (e.g. `DirA`) and may contain any combination of the following symbols:

- `.` (current directory)
- `..` (parent directory)
- `~` (home directory)

Command	Behavior
`cd` `cd ~`	Change the CWD to the home directory.
`cd DirA`	Change the CWD to the `DirA` directory in the CWD.
`cd ..`	Change the CWD to the parent directory.
`cd .`	Change the CWD to the current directory (no noticeable change)
`cd ../DirA`	Change the CWD to the `DirA` directory in the parent of the CWD.
`cd ~/DirA/DirB`	Change the CWD to `DirB`, which is in the `DirA` directory, which is in the home directory.

`mkdir [OPTIONS]... DIRECTORY...`

Creates new directories for each of `DIRECTORY`.

Command	Behavior
`mkdir DirA`	Create a new directory `DirA` in the CWD.
`mkdir ~/DirA/DirB`	Create the directory `DirB` in `DirA`, which is in the home directory.
`mkdir -p DirA/DirB`	Create the directory `DirB` in `DirA`, which may or may not exist. If it doesn't exist already, it will be created.

`rm [OPTION]... FILE...`

Deletes each of `FILE`. **Note that `rm` permanently deletes the file(s).**

Command	Behavior
`rm FileA`	Delete `FileA`.
`rm -r DirA`	Delete the directory `DirA`.
`rm -i FileA`	Delete `FileA`, but asks for confirmation before deleting.

```
cp [OPTION]...  SOURCE DEST
cp [OPTION]...  SOURCES...  DIRECTORY
```

Copies `SOURCE` into `DEST`. If multiple sources are provided, the destination must be a directory.

Command	Behavior
`cp FileA FileB`	Copy `FileA` into `FileB`.
`cp FileA FileB DirA`	Copy `FileA` and `FileB` into directory `DirA`.
`cp -r DirA DirB`	Copy `DirA` to `DirB`. Create `DirB` if it doesn't exist, otherwise copies `DirA` into `DirB`.

```
mv [OPTION]...  SOURCE DEST
mv [OPTION]...  SOURCES...  DIRECTORY
```

Moves `SOURCE` into `DEST`. If multiple sources are provided, the destination must be a directory.

Command	Behavior
`mv FileA FileB`	Rename `FileA` into `FileB`.
`mv FileA FileB DirA`	Move `FileA` and `FileB` into directory `DirA`.
`mv DirA DirB`	Move `DirA` to `DirB`. Rename `DirA` to `DirB` if `DirB` doesn't exist, otherwise move `DirA` into `DirB`.

```
find [PATH]...  [EXPRESSION]
```

Searches recursively in each of `PATH` with the constraints specified by `EXPRESSION`.

Command	Behavior
`find` `find .`	Output all the files and directories recursively in the CWD.
`find -type f`	Output all the files recursively in the CWD.
`find -type f -name '*File*'`	Output all the files recursively in the CWD that have the word `File` somewhere in their names.

File I/O and Processing

`echo [STRING]...`

Outputs each of `STRING` separated by new lines.

Command	Behavior
`echo 'Hello'`	Output 'Hello'.

`cat [OPTION]...  [FILE]...`

Outputs each of `FILE`. If no file is specified, uses `stdin`.

Command	Behavior
`cat FileA`	Outputs the content of `FileA`.

`less [OPTION]...  [FILE]...`

Outputs each of `FILE`. If no file is specified, uses `stdin`. The page is scrollable using the arrow keys. To escape, press `q`. To search, press `/` and then start typing the regular expression search query. To find the next instance of the search, press `n`, and to find the previous instance of the search, press `N`.

Command	Behavior
`less FileA`	Output the contents of `FileA` in a scrollable view.

`head/tail [OPTION]... [FILE]...`

Outputs either the beginning (`head`) or end (`tail`) of each of `FILE`. If no file is specified, uses `stdin`.

Command	Behavior
`head FileA`	Output the first 10 lines of `FileA`.
`tail -n 5 FileA`	Output the last 5 lines of `FileA`.

`tee [OPTION]... [FILE]...`

Simultaneously outputs `stdin` and copies it to each of `FILE`. It makes the most sense to use this command at the end of a series of pipes (e.g. `cat FileA | tee FileB` will output `FileA` and copy it to `FileB`).

Command	Behavior
`tee FileA`	Output `stdin` and copies it to `FileA`.

`tr [OPTION]... SET1 [SET2]`

Converts, squeezes, and/or deletes characters from `stdin` and outputs the result. Sets are strings of characters, where each character is treated independently. For example, 'hi' is not treating as the word 'hi', but rather as the individual characters 'h' and 'i'.

Command	Behavior
`tr -s ' '`	Squeeze multiple consecutive spaces into a single space.
`tr -d ' '`	Delete all spaces.
`tr 'ab' 'cd'`	Replace all 'a's with 'c's and all 'b's with 'd's.

`cut OPTION... [FILE]...`

Outputs parts of a line (e.g. columns, characters, etc.) for each line in each of `FILE`. If no file is specified, uses `stdin`. Ranges can be specified as `start-end`, where `start` and `end` are inclusive. Some valid ranges include:

- `1` (only column 1)
- `1-2` (columns 1 and 2)
- `-3` (all columns up to column 3)
- `4-` (all columns from column 4 to the end)

It may be useful to pipe a `tr` command into `cut` (e.g. `cat FileA | tr -s ' ' | cut -d' ' -f1-2`).

Command	**Behavior**
`cut -f1-2`	Output columns 1 and 2, where columns are delimited by a single `TAB` character.
`cut -d' ' -f4`	Output column 4, where columns are delimited by a single `SPACE` character.
`cut -c-15`	Output the first 15 characters of each line.

`grep [OPTION]... PATTERN [FILE]...`

Searches for the regular expression `PATTERN` in each of `FILE`. If no file is specified, uses `stdin`.

See C on page 44 for information on regular expressions.

Command	Behavior
`grep 'Hello' FileA`	Output all lines in `FileA` that contain the word 'Hello'.
`grep '[A-Za-z]\+' FileA`	Output all lines in `FileA` that contain at least one letter.
`grep '.*' FileA`	Output all lines in `FileA` (since all lines match `.*`).
`grep '[0-9]\{2\}/[0-9]\{2\}'`	Output all lines in `stdin` that contain a date of the format `mm/dd`. Note that this doesn't check for valid dates.
`grep '#include' *.cpp`	Output all the `#include`s in the `cpp` files in the current working directory.
`grep -r '#include' .`	Output all the `#include`s in all the files in the current working directory and all of the successive nested directories (recursive).

`sed [OPTION]... SCRIPT [FILE]...`

Executes the editing script `SCRIPT` on each of `FILE` and outputs the result. If no file is specified, uses `stdin`. `sed` scripts are composed of individual commands, which come in formats such as
`s/regex/replacement/modifiers` or `/regex/d`.

One example of a command is search and replace, denoted by `s`. The search and replace command matches each line with `regex` and substitutes each match with `replacement`. By default, the substitution happens once per line, but the `g` modifier can be used to substitute all matches on a line. A search and replace may look like `s/[0-9]\+/12345/g`, which replaces all decimal numbers with 12345.

Another example of a command is delete, denoted by `d`. The delete command will remove entire lines if any part of the line matches `regex`. A delete operation may look like `/^[A-Z]\+$/d`, which will delete any line that only contains upper case characters.

See C on page 44 for information on regular expressions and C on page 44 for information on `sed`.

Command	Behavior
`sed 's/Hello/Hi/g' FileA`	Output all lines in `FileA`, but replaces all instances of 'Hello' with 'Hi'.
`sed '/[A-Z][a-z]\+/d' FileA`	Output all lines in `FileA` other than those that contain a word that begins with a capital letter.
`sed 's/[A-Z][a-z]\+/DELETE/; /DELETE/d' FileA`	Same as above, but uses multiple commands (the above is more efficient).
`sed -n '/12345/p' FileA`	Output all lines in `FileA` that contain the number 12345 (behaves the same way as `grep`).

`awk [OPTION]... PROGRAM [FILE]...`

Executes the editing program `PROGRAM` on each of `FILE` and outputs the result. If no file is specified, uses `stdin`. `awk` programs are composed of individual actions, which come in the format `pattern { action }`. `pattern` may be a regular expression or some other built-in `awk` patterns (e.g. `BEGIN`). `action` is a C-like sequence of statements.

`awk` comes with some built in variables that can be used in the actions. One example is that `awk` automatically applies a delimeter (default delimieter is a space) and generates columns. These columns can be accessed using `$n`, where `n` is the column number. For example, `awk -F',' '/.*/ { print $2; }' FileA` will output the second column of `FileA`, where columns are delimited by commas (equivalent to `cut -d',' -f2 FileA`). `$0` is a special variable that is the entire line instead of a single column.

Additionally, since actions consist of a sequence of C-like statements, you can define your own variables. The following will output all lines of `FileA` except for the first, because `start` is initially equal to `1`: `awk 'BEGIN { start=1; }; /.*/ { if(start==0) { print $0; } else { start=0; } }' FileA`. `BEGIN` is a special pattern for which the action will be executed exactly one time before any lines are processed. It is useful for initializing variables, as shown in the example.

See C on page 44 for information on regular expressions and C on page 44 for information on `awk`.

Working Remotely

`ssh [OPTION]... [USER@]HOSTNAME`

Logs into the machine specified by `HOSTNAME` as `USER` and opens a shell.

Command	Behavior
`awk '/.*/ { printf "%s %s\n", $1, $2; }' FileA`	Output the 1st and 2nd columns of `FileA`.
`awk '/[0-9]+/ { printf "%s %s\n", $1, $2; }' FileA`	Output the 1st and 2nd columns of `FileA`, if the line contains a decimal number.
`awk '/.*/ { temp=$1; $1=$2; $1=temp; print $0; }' FileA`	Output `FileA`, but swaps the 1st and 2nd columns.

Command	Behavior
`ssh chirag@mario.ece.utexas.edu`	Log into `mario.ece.utexas.edu` as `chirag`.
`ssh -p2020 chirag@mario.ece.utexas.edu`	Log into `mario.ece.utexas.edu` as `chirag` on port 2020. The SSH server must be running on port 2020.

`scp [OPTION]... [[USER@]HOSTNAME1:]FILE1... [[USER@]HOSTNAME2:]FILE2`

Securely copies each of `FILE1` from `HOSTNAME1` to `HOSTNAME2` over SSH. If a hostname is not given, uses `localhost` (the local machine). The semantics are similar to the `cp` command. Note that the direction of the transfer is specified by the order of the parameters (always `FILE1...` to `FILE2`), and `scp` should be run on the local machine unless using reverse tunneling.

`vim [OPTION]... [FILE]...`

A fully terminal-based text editor. Operations are performed from one of the following modes:

- Normal mode (start off in this mode): used to type commands, usually through one or more keystrokes.
- Insert mode: used to type like in a GUI-based text editor.
- Replace mode: used to overwrite characters while typing, similar to pressing the insert key and then typing in a GUI-based text editor.
- Visual mode: used to make selections.
- Ex mode: used to enter commands that do not have keystrokes associated with them, which would normally be used in normal mode.

Command	Behavior
`scp FileA chirag@mario.ece.utexas.edu:~`	Copy `FileA` from the local machine to the home directory of `mario.ece.utexas.edu`.
`scp chirag@mario.ece.utexas.edu:~/FileA .`	Copy `FileA` from the home directory of `mario.ece.utexas.edu` to the current working directory of the local machine.
`scp -P2020 FileA chirag@mario.ece.utexas.edu:~`	Copy `FileA` from the local machine to the home directory of `mario.ece.utexas.edu` using port 2020.
`scp -r DirA chirag@mario.ece.utexas.edu:~`	Copy the entire contents of `DirA` from the local machine to the home directory of `mario.ece.utexas.edu`.

When in normal mode, you can type commands and motions. Commands can several things, including switching to insert mode, deleting text, or copying and pasting. Some commands can be combined with motions. Any action that moves the cursor is considered a motion. This includes moving up or down, moving to the next word, or moving up a page. For example, to delete a word, you can type `dw` from normal mode. To delete 3 words, you can type `d3w` from normal mode. Below are some common motions and commands.

Motion	Behavior	Motion	Behavior
`h`	Move left	`l`	Move right
`j`	Move down	`k`	Move up
`w`	Move to next word	`b`	Move to previous word
Command	**Behavior**	**Command**	**Behavior**
`i`	Enter insert mode	`c`	Delete and enter insert mode*
`d`	Delete*	`x`	Delete a single character
`R`	Enter replace mode	`r`	Replace a single character
`y`	Yank/copy*	`p`	Paste

*Requires motion

See C on page 44 for information on `vim`.

`tmux [COMMAND [PARAMS]]`

Creates a multiplexed terminal session (i.e. multiple panes and windows). The session is created as an independent process instead of being a subprocess of the shell. In other words, even if the shell is terminated, the tmux session will persist. This is extremely useful when SSHing into a remote machine, as you can disconnect the SSH session and then resume the tmux session at a later time (or even from a different client machine). To create a new session, run `tmux new -s session-name`. To reattach to an existing session, run `tmux attach -t session-name`. To exit a pane or window, run `exit` from within that pane or window. Once all panes and windows in a tmux session are closed, the tmux session is terminated.

When in a tmux session, all commands to manage the session begin with a prefix keystroke. By default, the prefix is `ctrl+b`. Note that for tmux, you press the prefix keystroke, then release all keys on the keyboard, and then type in the command keystroke. The following are common commands in a tmux session. Each one should begin with the prefix keystroke.

tmux Command	**Behavior**
`d`	Detach from the session (i.e. leave the session without terminating it).
`c`	Create a new window.
`n`	Move to the next window.
`p`	Move to the previous window.
`number`	Move to the window labeled `number`.
`%`	Create a vertical pane.
`"`	Create a horizontal pane.
`arrow`	Move to the pane in the direction of the arrow pressed.

See C on page 44 for more information on `tmux`.

Managing Files and Packages

`zip [OPTION]... [ZIPFILE [FILE]...]`

Creates a zip archive with each of `FILE` with the name `ZIPFILE`.

`unzip [OPTION]... ZIPFILE`

Extracts the zip archive `ZIPFILE`.

Command	Behavior
`zip files.zip FileA FileB`	Create the zip archive `files.zip` out of `FileA` and `FileB`.
`zip -r files.zip DirA`	Create the zip archive `files.zip` out of directory `DirA`.

Command	Behavior
`unzip files.zip`	Extract the zip archive `files.zip` in the current working directory.
`unzip -d OutDir files.zip`	Extract the zip archive `files.zip` into `OutDir`.

`tar [OPTION]... ARCHIVE`

Creates or extracts the tarball `ARCHIVE`. A tarball is a single file concatenation of all the files in the archive. Optionally, the `tar` command can run `gzip` to compress the concatenated file.

Command	Behavior
`tar -cf files.tar *`	Create the tarball `files.tar` of all the files (recursively) in the current working directory.
`tar -xf files.tar`	Extract the tarball `files.tar` in the current working directory.
`tar -czf files.tar.gz *`	Create and gzip the tarball `files.tar.gz` of all the files (recursively) in the current working directory.
`tar -xzf files.tar`	Decompress and extract the gzipped tarball `files.tar.gz` in the current working directory.

`chmod [OPTION] MODE FILE...`

Applies the permission string `MODE` to each of `FILE`. The permission string can be given in one of two formats.

The first format treats permissions as a 3-digit octal number, where the most significant digit represents the permissions for the owner, the middle digit represents the permissions for the group the file belongs to, and the least significant digit represents the permissions for the other users (not the owner and not in the same group as the file). For example,

`chmod 777 FileA` will grant read, write, and execute permissions to all users, and `chmod 600 FileA` will grant read and write permissions to the owner and prevent access to any other users.

The second format is more legible, but also more verbose. It consists of specifying a tier to modify (`u` for owner, `g` for group, `o` for others, and `a` for all users), followed by a `+` to add permissions and `-` to remove permissions, followed by the permissions to grant (`r` for read, `w` for write, and `x` for execute). Multiple mode changes can be separated with `,` such as `u+w,o-w`.

Command	Behavior
`chmod 700 FileA`	Grant all permissions to the owner of `FileA` and revoke all permissions from everyone else.
`chmod u+rwx,go-rwx FileA`	Same as above.
`chmod +x FileA`	Grant execute permissions to all tiers of permissions.
`chmod 500 DirA`	Grant read and execute permissions to the owner of `DirA` and revoke all permissions from everyone else.
`chmod -R u+rw DirA`	Grant read and write privileges to the owner of `DirA` for all files and directories in `DirA`.

`sudo COMMAND`

Runs `COMMAND` as the superuser (also called `root`).

Command	Behavior
`sudo ls /root`	View the files in the `root` user's home directory.
`sudo -s`	Open an interactive shell as the `root` user.

`apt SUBCOMMAND [PARAMS]...`

`apt` is the package manager on Debian-based distributions of Linux. `apt` provides a set of subcommands to manage packages.

`apt install PACKAGE...`

Installs each of `PACKAGE` from the repositories. Requires superuser permissions.

Command	Behavior
`apt install vim`	Install `vim`.

`apt remove PACKAGE...`

Uninstalls each of `PACKAGE`. Requires superuser permissions. After running `apt remove`, it may be useful to run `apt autoremove`, which will remove any packages that are no longer necessary (i.e. dependencies that were installed only for the purpose of being able to run one or more of `PACKAGE`).

Command	Behavior
`apt remove vim`	Uninstall `vim`.

`apt upgrade`

Upgrades all packages installed on the system, including kernel packages. Requires superuser permissions.

Command	Behavior
`apt upgrade`	Upgrade all packages on the system.

`apt update`

Updates the local (cached) copy of the software available in the repositories. Requires superuser permissions.

Command	Behavior
`apt update`	Update cached package listing.

`make [OPTIONS]... TARGET`

Builds `TARGET`, which is specified in the `Makefile`. `make` is often preceded by a command or series of commands that will generate a `Makefile`. When using the GNU autotools build system (which many open source projects employ), the command is often `./configure`.

`configure` an be given parameters that will generate a custom `Makefile`. A common parameter is `--prefix=/absolute/path/to/prefix/`, which specifies where the installation build target (usually `make install`) will install the project. Other parameters may be project specific to enable or disable features, such as `--enable-ssl`.

`make` can also be given parameters that will either be used by the `make` command or passed into the source files of the project. For example, the `-jN` flag can be used to specify up to `N` threads can be used during the build (increasing parallelism, and thus increasing speed of the build). The `-D` flag can be used to define variables that the compiler will see, such as `-DENABLE_DEBUG=1`, which is the equivalent of putting `#define ENABLE_DEBUG 1` at the top of all C/C++ files.

Command	Behavior
`make`	Execute the default target in the `Makefile`.
`make -j4`	Same as above, but enable up to 4 concurrent threads.
`make install`	Execute the `install` target in the `Makefile`.
`make -f BUILDME`	Execute the default target in the Makefile `BUILDME`.
`make -DDEBUG=1`	Execute the default target in the `Makefile` and effectively adds `#define -DDEBUG=1` during compilation (for a C/C++ project).

Useful References

Regular Expressions

- Tutorial
- Debugger

`sed`

- Tutorial

`awk`

- Tutorial

`vim`

- The `vimtutor` command
- Vim Adventures (tutorial game)
- Quick reference sheet

`tmux`

- Tutorial
- Quick reference sheet

Appendix D

Lecture Slides

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

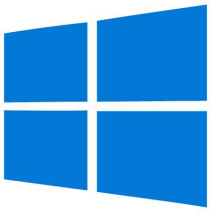
Basics of Linux

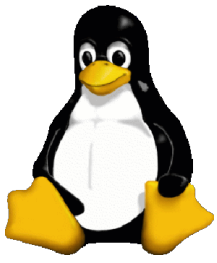
EE 107S: Introduction to Linux

Lecture 1: 01/17/17

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

What is Linux?





TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Why Linux?

- Very likely to encounter Linux at an internship or job
- Makes software development easier
- Almost 100% customizable (both a blessing and a curse)
- Well supported by a massive community
- Updates that are easy to install and completely under your control
- More secure than Windows
- More stable than Windows
- Free (usually open source) alternatives to almost everything
- Works on pretty much any hardware
- …

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

How do we get Linux?

- A computer can only run one operating system (OS) at a time
- We can create a “virtual” machine to run Linux “virtually”
  - A common way of running multiple OSes at the same time

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

How do we get Linux?

- Ubuntu 16.04.1 LTS
- VirtualBox (latest)
- Video Tutorial

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

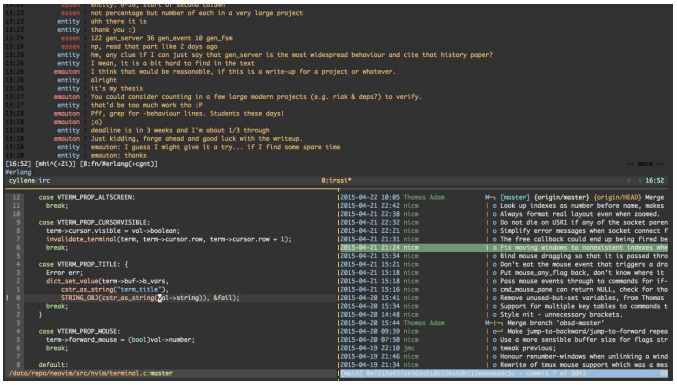
Alternative ways of using Linux

- Install Linux as a primary OS
- Log into remote machine that runs Linux

Basics of a terminal

- Text-only interface
- Enter ”commands” (really, programs) into the terminal to perform actions
  - Commands accept flags and parameters
- Concept of a current working directory (CWD)

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD



What is going on??

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Our first command: `ls`

- Many commands operate on files or directories
- Shows the contents of the current working directory
- Directories show up in a different color

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Man's best friend: `man`

- "man pages" are the owner's manuals for every command
- When in doubt, type in `` `man <command>` `` for a description of what a command can do
- Take note of the … next to [OPTION] and [FILE]

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Where are we?: `pwd`

- Lists the full path of the CWD
- Nested directories paths are separated by /

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Navigating directories: `cd`

- Changes the CWD
- Move to the parent directory using ..
- Current directory is .
- Home directory is ~

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Creating a new directory: `mkdir`

- Only parameter is the name of the directory
- Names can be complex (i.e. contain ., .., etc.)

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Deleting files/directories: `rm`

- Parameter is file or directory to delete
- If directory, must use `-r` flag
- NO RECYCLE BIN OR TRASH WHEN USING `rm`

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Copying files/directories: `cp`

- Similar to `rm`, but takes source and destination parameters
- `-r` flag is necessary to copy directories

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Moving files/directories: `mv`

- Similar to `cp` with source and destination
- `-r` flag NOT necessary to move directories

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Practice

- Make the following directories:
  - `~/lecture1/dir1/dirA`
  - `~/lecture1/dir1/dirB`
  - `~/lecture1/dir2/dirA`
  - `~/lecture1/dir2/dirB`

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Possible Solution

```
cd ~
mkdir lecture1
cd lecture1
mkdir dir1
cd dir1
mkdir dirA dirB
cd ..
cp -r dir1/* dir2
```

Possible Solution

```
cd ~
mkdir -p lecture1/dir1/dirA lecture1/dir1/dirB
cp -r lecture1/dir1 lecture1/dir2
```

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Practice

- Given the previous directories, make the following directories:
  - `~/lecture1/dir2/dir1/dirA`
  - `~/lecture1/dir2/dirA`
  - `~/lecture1/dir2/dirB`

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Possible Solution

```
cd ~
mv lecture1/dir1 lecture1/dir2
rm -r lecture1/dir2/dir1/dirB
```

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Advanced searches: `find`

- Lists all files/directories at the path given as a parameter (including nested files/directories)
- Specify whether to show just files or directories with `` `-type f` `` or `` `-type d` ``, respectively

Useful shortcuts

- Up/down keys to bring back previous commands
- Ctrl+R to search previous commands
- <TAB> for autocompletion!

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Lecture and assignment "recordings"

- `script -t -a session-name 2> timing-name`
  - `session-name` and `timing-name` are filenames
- `scriptreplay timing-name session-name`

Assignment

- Come to lab section tomorrow or Thursday
  - Only session we will require attendance for

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

File I/O and Processing

EE 107S: Introduction to Linux

Lecture 2: 01/24/17

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Announcements

- Piazza
- Office hours
  - Chirag: Th 2:00 – 3:30 PM in AHG lounge
  - Cassidy: T 5:30 – 6:30 PM in AHG at HKN table

Wildcards

- The * character matches anything in the CWD
  - Given files `file1` and `file2`: `file*` would match both names
- * can be used for directories too
  - Given files `dir1/fileA` and `dir2/fileA`: `dir*/fileA` would match both files

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Searching for files/dirs: `find`

- Use `-name` argument with wildcards to search for files/directories
  - `find . -name 'fullname' -type d`
  - `find . -name '*part*' -type f`

Display input as output: `echo`

- `echo` takes a string argument and outputs it
  - `echo 'Hello, world!'`
- Doesn't sound very useful, but it can be…we'll use it soon

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Viewing files: `cat`

- Accepts a parameter of a file name
  - This parameter is optional, in which case it can accept input from `stdin` (more on this later)
- Outputs the contents of the file

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Viewing files: `less`

- Works similar to `cat`, but has a scrollable view
- Very useful when displaying lots of output
- Press ‘q’ to exit, like `man`

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Viewing parts of a file: `head`/`tail`

- `head` outputs the first 10 lines of a file
- `tail` outputs the last 10 lines of a file
- Both commands accept the `-n` flag, which specifies the number of lines to show

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

I/O == `stdin`/`stdout`

- Input is captured from 'standard input' (`stdin`)
- Output is displayed to 'standard output' (`stdout`)
- Input and output can be redirected
  - Files can be fed into input, instead of typing them
  - Outputs can be sent to a file instead of to the terminal

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Redirecting output

- The output of a command can be sent to a file by adding the '>' character and then specifying a file name at the end of the command
  - `ls -l > files.txt`
- '>>' can be used to append to an output file
  - `echo 'End of file' >> files.txt`

Redirecting input

- A file can be used as input instead of the keyboard by adding the '<' character and then specifying a file name at the end of the command
  - `cat < files.txt` (same result as `cat files.txt`)
  - `less < files.txt` (same result as `less files.txt`)

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Pipes

- Takes the output of one command and sends it to the input of the next
- Allows you to chain commands together
  - May not be immediately obvious, but this is extremely powerful
- Separate commands with the `|` character
  - `cat files.txt | less`

Viewing and saving outputs: `tee`

- Accepts input through `stdin` and then displays it back
- Takes a file argument and copies the output to the file
- Useful if you want to save output while watching it, such as if a command produces a lot of output
- Can pipe output into `less` as well

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Redirection exercise

1. Use `tee` to display `/proc/cpuinfo` and write the output to the file `cpus`, simultaneously
2. Output the file to `stdout`

Possible solution

```
cat /proc/cpuinfo | tee cpus | less
cat < cpus
```

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Intro to text processing: `tr`

- Translate one character to another
- Delete characters
- Compress consecutive characters into a single character

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Delimeters

- A character that separates parts of a line
- Common delimeters are ‘ ‘ and ‘,’
- Many of the text processing tools allow you to specify alternate delimeters

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Bonus command: `curl`

Used to interact with webpages

```
curl http://a-dev.me/107s/gradebook.txt > gradebook.txt
```

Splitting by columns: `cut`

- Separates each line into columns by delimeter
  - Default delimeter is tab character
- Accepts a range of columns to output
  - Given as '1-3', '-3', '3-', etc.
  - Ranges are inclusive, column numbers start at 1

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Regular expressions

- A more flexible way of matching strings
  - Allow a lot more control than wildcards
- '[0-9]+' matches a string of one or more numbers
  - 'ab0123cde' and '4ef' match, but 'abcde' doesn't
- '[A-z]*' matches a string of zero or more letters (upper case and lower case)
  - 'abCdef1234', '1bc', and '1234' all match
    - Why does 1234 match? Because * means **zero** or more letters

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Regular expressions

- Structured as 'character classes' followed by number of instances
  - Character classes can be in square braces like [0-9]
  - Any character in the range matches, regardless of order
- Number of instances given by +, *, ?, etc.
  - +: one or more
  - *: zero or more
  - ?: exactly zero or one

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Regular expressions

- . will match any single character
  - Therefore .* will match every single line
- Character classes can be combined together into a large regular expression

Regular Expression examples

- Date: [0-9][0-9]/[0-9][0-9]/[0-9]{2}
- Hex number: x?[a-fA-F0-9]+
- Email address: [A-z0-9]+@[a-z]+\.com

Filtering output: `grep`

- Reads input (file or `stdin`) line by line
- Extracts lines that match regular expression
- Useful for searching through many files

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

‘Stream’ editing: `sed`

- Reads input (file or `stdin`) line by line
- Modifies the lines (make changes, delete, etc.)
- Syntax is given as ‘operation/regex/modifiers’

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Pretty much everything else: `awk`

- Reads input (file or `stdin`) line by line
- Format of an awk operation is ‘`regex1 { operation }; regex2 { operation } ...`’
- Common operation is `print`
  - There are special variables `$0`, `$1`, … that can be used for columns
- The operations are similar to C code, which means you can have variables!

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

File processing exercise

Swap the first name and last name columns in the `gradebook.txt` file using awk

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Possible solution

```
awk '/.*/ {temp=$1; $1=$2; $2=temp; print $0;}' gradebook.txt
```

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Putting it all together: Bash Scripts

- Bash scripts (different than the `script` command) contain a sequence of steps
- When the script is run, it executes all the commands as if it were a single command
- Bash is a full programming language, but we'll get to that later

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Assignment

- Will be posted some time tonight
- It will exercise your text processing skills

Tutorials

- Regular expressions
  - Regular expression 'debugger'
- sed
- awk

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Working Remotely

EE 107S: Introduction to Linux

Lecture 3: 01/31/17

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Logging in remotely: `ssh`

- Specify a remote machine to log in to
- Log in using username and password for remote machine
- Can set up special files to allow easier access (SSH keys)

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

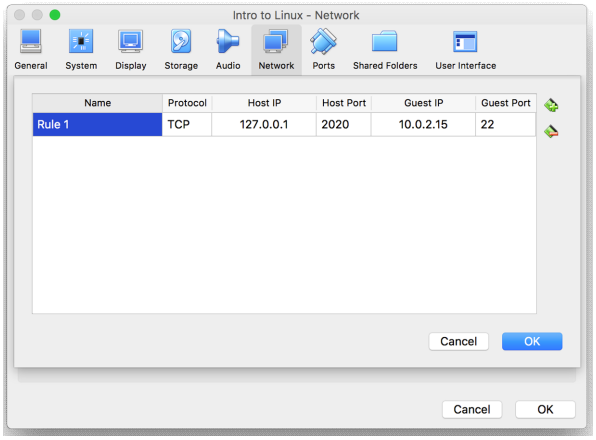
SSH Clients

- Windows: MobaXterm
- macOS: built-in terminal

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Setting up SSH with VirtualBox

```
sudo apt install openssh-server
ssh chirag@127.0.0.1
ip addr | grep inet
```



Setting up SSH with VirtualBox

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Transferring files: `scp`

- Arguments are the same as `cp` (`src dest`)
  - Remote machine is specified as:
    `chirag@127.0.0.1:~/path/to/file`
- `scp` should be run from local machine, regardless of direction of transfer

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

The best text editor: `vim`

- Fully terminal-based (editing without a mouse?!?)
- Available on most fresh installations of Linux distributions (maybe just `vi`)
  - `sudo apt install vim`

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Vim modes

- Normal mode (start off in this mode)
- Insert mode
- Replace mode
- Visual mode
- Ex mode

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

The Vim mindset

- Commands perform an action
- Motions move the cursor
- In normal mode, enter a series of commands
  - Some commands must be followed by a motion
- It's not really easy to learn Vim, but once you do you'll be a better person

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Common commands

- `i` - enter insert mode
- `d` - delete (requires motion)
- `x` - delete a single character
- `r` - replace a single character
- `c` - change (requires motion)
- `y` - yank/copy (requires motion)
- `p` - paste

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Common motions

- `w` - move forward a word
- `b` - move back a word
- Preface a motion with a number
  - `3w` - move forward 3 words

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

What's going on?

- `ggdG` - delete all the lines in a file
- `0c$` - change the line (also `cc`)
- `di{` - delete everything within curly braces
- `87(ctrl+a)` - increase the next number by 87
- `:%!sort` - sort all the lines in a file

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Most basic usage

- `i` - insert mode
- `<ESC>` - return to normal mode
- `:wq` - save and quit

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Navigating in normal mode

- h - left one character
- `j` - down one line
- k - up one line
- `l` - right one character
- The idea is that you don't need to move your fingers away from the home row to the arrows

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Spicing it up

- From normal mode, `d` is a delete command
  - You need to specify a motion for the delete command (i.e. what do you want to delete?)
  - `dj` - delete from the current line and then down one line
- From normal mode, `x` deletes a single character (the same as dl)

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

More insert modes

- Press `a` to start insert mode one character to the right of the cursor
  - What's the point? You could do `l` then `i`. Efficiency.
- Press `A` to move to the end of the line and go into insert mode in one command
- Press `I` to move to the start of the line and go into insert mode in one command

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

History

- u - undo
- `ctrl+r` - redo

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Searching and replacing

- `/` - puts you into search mode (just start typing a regex)
  - Case sensitive by default
  - n to go to next match, N to go to previous match
- `:%s/regex/replacement/g` - search and replace
  - Similar to `sed`

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Useful resources

- `vimtutor` command
- http://vim-adventures.com/
- Command cheat sheet

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Terminal multiplexer: `tmux`

- Provides many features, some of which are
  - Multiple windows/panes in one session
  - Persistent sessions (even if you disconnect SSH)
  - Simple collaboration
- As if Vim wasn't hard enough, here's another thousand keyboard shortcuts

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Tmux operations

- Operations start with a prefix (default `ctrl+b`)
- Press prefix, then a keyboard shortcut for an operation
  - Operations include creating a new window, entering a tmux command, etc.

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Windows

- `ctrl+b, c` - create new window
- `ctrl+b, n` - move to next window
- `ctrl+b, p` - move to previous window
- `ctrl+b, number` - move to window number

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Panes

- `ctrl+b, "` - new horizontal pane
- `ctrl+b, %` - new vertical pane
- `ctrl+b, arrows` - move to different pane

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Managing sessions

- ctrl+b, d - detach session (i.e. hide tmux)
- `tmux new -s session-name`
- `tmux attach -t session-name`

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Managing Files and Software

EE 107S: Introduction to Linux

Lecture 4: 02/07/17

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

File archives: `zip`

- First parameter is archive name, remaining parameters are files/directories
- Create a zip archive from multiple files
- `-r` can be used for directories
- `unzip` extracts zip archive

File archives: `tar`

- Tarball is an uncompressed concatenation of multiple files
- Typically using with `gzip` to compress
  - `gzip` can only compress a single file

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Tiers of permissions

1. Permissions for the owner of the file
2. Permissions for the group the file belongs to
   - Groups are composed of users
3. Permissions for the others (not the owner and not in the group the file belongs to)

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Types of permissions

- Read: content of files can be read
- Write: file content can be changed
- Execute: file can be executed as a script
  - To `cd` to a directory, that directory must have execute permissions

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Reading permissions

- Using `ls -l` command:

```
drwxrwxr-x 2 chirag chirag ... lec1
-rw-rw-r-- 1 chirag chirag ... FileA
```

- Leftmost column is string of permissions; 3 letters for each of user, group, and others
  - First letter represents directory or file
  - Each set of three letters represents read (r), write (w), or execute (x)
- Following two columns are owner and group that the file belongs to

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Changing permissions: `chmod`

- Treat permissions field as 3 digit octal number
  - `rwxrwxrwx` = 777 (all users can read, write, or execute)
  - `rw-r-----` = 640 (owner can read/write, group can read, others can't access)
- First parameter is new permissions to assign, parameters after that are files to change
- Use `-R` to recursively change

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Alternate syntax

- Specify tier with user (u), group (g), other (o)
- Specify whether to add (+) or subtract (-)
- Specify permission (r, w, x)
- Examples
  - ug+x: add execute permission to user and group
  - o-w: remove write permission for others

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Exercise

1. Create a file `temp` in your home directory and see what the default permissions are
2. Remove all group and other permissions
   – Very important if you're doing group projects on a shared machine!

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Possible solution

```
touch file
ls -l file
chmod go-rwx file
```

Possible solution

```
touch file
ls -l file
chmod 500 file
```

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Superuser privileges: sudo

- sudo and then a command
  - Runs the command as a superuser
- Superuser privileges supersede all file permissions (i.e. can read, write, execute all normal files)

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Exercise

- Create a file in `/root` (home directory of the superuser) and see the default permissions are
- Change the permissions so that any user can modify the file

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Attempt #1

```
sudo touch /root/file
sudo ls -l /root/file
sudo chmod o+w
echo "hi" >> /root/file
```

Possible solution

```
sudo touch /root/file
sudo ls -l /root/file
sudo chmod o+w /root/file
sudo chmod o+rw /root
echo "hi" >> /root/file
```

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Managing packages: apt

- Think of repositories like an app store
- Used to add/remove and update software
- Must be superuser to change software
  - apt has some features that don't require superuser

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Adding software: `apt install`

- Parameters are list of packages to install
- Automatically installs all necessary dependencies

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Removing software: `apt remove`

- Parameters are list of packages to remove
- Only removes packages, but residual unused dependencies are marked as “removable”
  - `apt autoremove`

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Upgrading software: `apt upgrade`

- One command away from updating all software on a machine!
- Includes kernel upgrades
- Upgrades are done in place, which means no restarting (unless it's a system update)

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Updating repository: `apt update`

- Packages are listed in a live repository (i.e. website), which is cached locally
- Sometimes the cache gets out of date, so you need to synchronize it with the repository
  - If out of date, you might get strange errors like "No packages found"

Where is software installed?

- `/usr/bin`: binary executables
- `/usr/lib`: shared libraries
- `/usr/include`: header files
- May be some alternative prefix (e.g. `/usr/local/*`)

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Build systems

- Whenever working on or creating a new project, you need some way to generate the executable
- This process is called building, and there are many tools that can help you make building easier

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

`make`

- Looks for a file called `Makefile`
- `Makefile` specifies build targets and a "script" for how to compile the target
  - Targets may include "Release", "Debug", "install"
- Can specify flags to `make`
  - `-j` enables parallel builds (e.g. `-j4` uses 4 threads)
  - `-D` adds C++ #define (e.g. `-DENABLE_FEATURE_A=1`)

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

`configure`

- Projects will often come with a configure script that creates a `Makefile`
- Configure script can set up `Makefile` with certain flags
  - One flag is `--prefix`, which specifies where `make install` should copy the files

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Installing `tmux` from source

```
git clone https://github.com/tmux/tmux.git
mkdir ~/sw
cd tmux
sh autogen.sh
./configure --prefix=/home/chirag/sw
make -j4
make install
~/sw/tmux
```

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Resources

- Advanced file permissions
- Writing a Makefile (we'll learn the basics in lab)
- Build systems for C++

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Version Control

EE 107S: Introduction to Linux

Lecture 5: 02/14/17

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

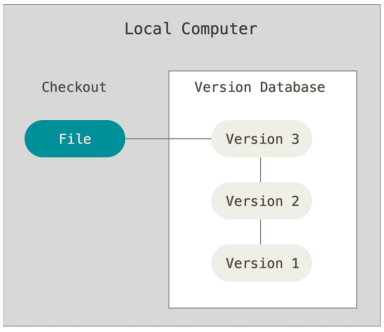
What is version control?

- A sophisticated backup system (but much more)
  - Maintains a record of changes made to a file
- Common examples: Dropbox, OneDrive, making several copies of a file, emailing yourself, etc.
- Probably every company that maintains code will use version control

Why version control?

- You don't want to lose your work
- Keep track of different versions and easily switch between them
- Keep a remote copy of files and their versions
- **Collaboration is much easier**

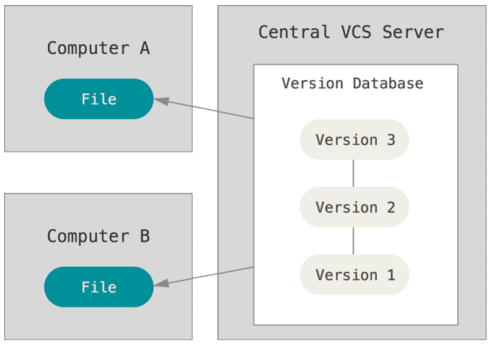
TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD



The most basic form of version control

Photo courtesy of git-scm
https://git-scm.com/book/en/v2/book/01-introduction/images/local.png

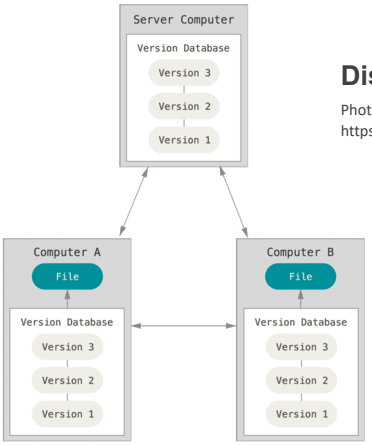
TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD



Centralized version control

Photo courtesy of git-scm
https://git-scm.com/book/en/v2/book/01-introduction/images/centralized.png

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD



Distributed version control

Photo courtesy of git-scm
https://git-scm.com/book/en/v2/book/01-introduction/images/distributed.png

Common version control systems

- Centralized
  - Subversion (SVN)
  - Perforce
- Distributed
  - Git
  - Mercurial

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Git

- Created by Linus Torvalds (creator of Linux)
- Treats versions as a DAG
- A set of tools that helps you navigate and modify the DAG

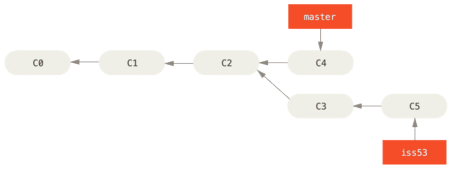


Photo courtesy of git-scm
https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Git

- Different ”states” a file can be in
  - Untracked
  - Modified
  - Staged
  - Committed (local)
  - Pushed (remote)
- Each version is called a commit
- Concept of a HEAD (where you are in the DAG currently)
- Branches in the DAG correspond to branches in `git`

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Popular remote providers

GitHub

Atlassian Bitbucket

Typically costs money for private repositories, but you can get unlimited repositories for free as a student. Take advantage of it!

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Github account and SSH keys

- Generate SSH keys (so you don't have to use a password)
  - `ssh-keygen -t rsa –b 4096`
- Create Github account and add SSH key

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Configuring Git

- Set your name
  - `git config --global user.name "Chirag Sakhuja"`
- Set your email address (the one with the Github account)
  - `git config --global user.email "chirag.sakhuja@utexas.edu"`

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Git status

- Gives you useful information on the status of your repository
- Often gives you verbatim commands to run
- Good practice to `git status` all the time

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Create a local repository

```
git init
vim README
```

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Creating a commit

```
git add README
git commit
```

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Pushing to a remote

```
git remote add origin
     git@github.com:chiragsakhuja/intro-to-
     linux.git
git push -u origin master
```

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Cloning an existing repository

```
git clone git@github.com:chiragsakhuja/intro-
    to-linux.git
```

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

References

- Cheat sheet
- Tutorials
- Documentation

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Process Management

EE 107S: Introduction to Linux

Lecture 6: 02/21/17

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Background and foreground processes

- Before UIs, we needed to be able to run multiple processes concurrently
- So far everything we have run is a foreground process
- Can have up to one process in the foreground and as many in the background
- To run a command in the background, add & at the end

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Seeing running processes: `jobs`

- Lists all background processes
- Either `running` or `suspended`
  - Processes can be suspended with `ctrl+z`
- Each process is given an ID

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Bringing back processes: `fg`

- Picks the item in `jobs` with a + next to it to resume
- Job ID can be specified with %
  - `fg %2`

Resuming to the background: bg

- Works the same way as `fg`

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Listing processes: ps

- Default behavior shows processes in current terminal
- Add -a to see other users' processes
- Add -x for more information
- Add -u to specify a user

Task manager: `htop`

- Output like a task manager
- Contains information on CPU/memory usage
- Can be displayed in tree form
- Can be used to kill processes

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Terminating processes: `kill`

- Sends a “signal” to the process specified by PID
- Common signal is 9 for SIGKILL
  - `kill -9 12345`

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Environment variables

- Variables set by the shell for all commands to access
- PATH keeps track of where executables are stored
- HOME has the same path is ~
- Can access environment variables with `echo`
  - `echo $PATH`
- Can set environment variables using `export` command (works in `.bashrc` too)

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Crunchbang/shebang

- Must be first line in the file
- Tells the shell how it should execute the file
- `#!/bin/bash` at the top of `script.sh` means run the command:

```
/bin/bash script.sh
```

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Variables

- Name followed by = followed by value (no spaces)
  - Value can be a number, a string, or the output of a command
    - `num=1`
    - `str='hello'`
    - `out=$(ls -l)`
- Refer to variables with `$varname`

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

Command line arguments

- Arguments are stored in special variables with `$number`
  - `$0` is the script name
  - Remaining arguments are 1-9 (no more than that as far as I know)
- $# contains number of arguments

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

If statements

```
if [[ condition ]]
then
  echo ”You made it”
else
  echo ”You didn’t make it :(”
fi
```

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

If statements

- Condition can be composed of various checks, some of which include
  - `INTEGER -eq INTEGER`
  - `STRING = STRING`
  - `-e FILENAME`
- Conditions can be combined with && or ||

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

While loops

```
counter=1
while [[ $counter -le 10 ]]
do
    echo $counter
    counter=$(($counter + 1))
done
```

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

For loops

```
for file in $(find . -type f)
do
    if [[ -e $file ]]
    then
        echo ”$file exists”
    fi
done
```

Reminder

- Near the end of the semester we'll send out an eCIS
- Please fill it out!

TEXAS
The University of Texas at Austin
WHAT STARTS HERE CHANGES THE WORLD

References

- Bash scripting tutorial

Bibliography

[1] Bruce Barnett. Awk - a tutorial and introduction. `http://www.grymoire.com/Unix/Awk.html`, 2016.

[2] Bruce Barnett. Sed - an introduction and tutorial. `http://www.grymoire.com/Unix/Sed.html`, 2016.

[3] Firas Dib. Online regex tester and debugger. `https://regex101.com`, 2017.

[4] Doron Linder. Vim adventures - learn vim while playing a game. `http://vim-adventures.com`, 2012.

[5] Daniel Miessler. A tmux primer. `https://danielmiessler.com/study/tmux/`, 2013.

[6] Richard Petersen. *Linux: The Complete Reference; Sixth edition.* McGraw-Hill, 1 edition, 2008.

[7] Regexone - learn regular expressions. `https://regexone.com/`, 2016.

[8] The linux documentation project. `http://www.tldp.org/`, 1999.

[9] Tmux cheat sheet & quick reference. `https://tmuxcheatsheet.com/`, 2017.

[10] Vim cheat sheet - english. `https://vim.rtorr.com/`, 2017.